FINAL YEAR PROJECT REPORT

EGOCENTRIC IMAGE CAPTIONING

By

A/C ABD US SAMMI OZAM

Pak/20095011, 95(B) EC

ADVISOR

WING COMMANDER NAYYER AAFAQ

CO-ADVISOR

ASSISTANT PROFESSOR DR IJAZ KHAN

COLLEGE OF AERONAUTICAL

ENGINEERING

PAF Academy, Asghar Khan, Risalpur

February 13, 2024

EGOCENTRIC IMAGE CAPTIONING

By

A/C ABD US SAMMI OZAM

Pak/20095011, 95(B) EC

ADVISOR

WING COMMANDER NAYYER AAFAQ

CO-ADVISOR

ASSISTANT PROFESSOR DR IJAZ KHAN

Report submitted in partial fulfillment of the requirements for the degree of Bachelors of Engineering in Avionics, (BE Avionics)

In

COLLEGE OF AERONAUTICAL ENGINEERING

PAF Academy, Asghar Khan, Risalpur

February 13, 2024

Approval

It is certified that the contents and form of the project entitled "Egocentric Image Captioning" submitted by Aviation Cadet Abd Us Sammi Ozam have been found satisfactory for the requirement of the degree.

Advisor: Wg Cdr Dr. Nayyer Aafaq

Signature: ________________

Date: ________________

Co-Advisor: Asst Prof Ijaz Khan

Signature: ________________

Date: ________________

Dedication

I want to take this opportunity to express my sincere gratitude to all those who have played a significant role in making this project possible. I would like to extend a special thanks to my loving family and my dedicated advisor, whose unwavering support and guidance have been instrumental in my academic success. Without their encouragement and support, I would not have been able to reach this point. To my parents, in particular, I owe a debt of gratitude for their selfless sacrifices, endless encouragement, and unwavering belief in me. This report is a tribute to them and all those who have contributed to my journey, and I am honored to share this achievement with them.

Acknowledgement

I express my sincere gratitude to Allah Almighty, who bestowed upon me the strength and determination to complete this project to the best of my abilities. My parents, whose unfailing love, unrelenting support, and steadfast prayers have been the compass in my life, deserve the utmost gratitude. Without their support, this accomplishment would not have been possible. I am immensely grateful to my advisor, Wg Cdr Dr. Nayyer Aafaq, for his constant guidance, invaluable feedback, and unwavering support. His encouragement and mentorship have been instrumental in shaping my research skills and intellectual growth. I also extend my heartfelt thanks to my co-advisor, Dr. Ijaz Khan, for his valuable input and for sharing his expertise in the field. I want to acknowledge the invaluable assistance of Lab Engineer Zahoor, Avn Cdt Abdullah Nawaz, and Avn Cdt Aqib, who provided me with their support and help whenever I needed it the most. I am grateful to all my teachers and colleagues who have contributed to my academic and professional growth in various ways. Finally, I would like to acknowledge the support of my friends and family members, who have been there for me throughout my academic journey. Thank you all for your support, encouragement, and motivation.

Abstract

Egocentric image captioning is the process of automatically generating descriptive text for images captured from a first-person perspective. This specialized task involves leveraging techniques from deep learning, computer vision, and natural language processing to build a model capable of comprehending the wearer's viewpoint and articulating it in a human-like language. The primary objective of egocentric image captioning is to offer an alternative means for human-machine interaction, especially catering to the visually impaired. Additionally, it aims to enhance the performance of image search engines by providing more contextually relevant and personalized results based on the user's perspective. In recent years, advancements in deep learning have significantly propelled egocentric image captioning into the forefront of research. Various approaches and evaluation metrics are actively explored to improve the accuracy and diversity of generated captions. Unlike general image captioning, egocentric image captioning demands a nuanced understanding of the wearer's visual experiences and the ability to articulate coherent and contextually relevant sentences. Typically, egocentric image captioning models employ a combination of Convolutional Neural Networks (CNNs) to extract visual features from the wearer's point of view. Recurrent Neural Networks (RNNs) or transformers are then employed to generate captions that reflect the wearer's interactions and observations. Addressing challenges such as ensuring both accuracy and diversity in generated captions is crucial. Researchers have introduced techniques like attention mechanisms to enhance the quality and variety of generated egocentric captions. The evaluation of egocentric image captioning models involves standard automatic metrics such as BLEU 1-4, METEOR, ROUGE, and CIDEr. These metrics gauge the similarity between the generated captions and ground-truth captions provided by human annotators. However, considering the unique perspective of egocentric data, human evaluation remains vital to comprehensively assess the quality of generated captions. Egocentric image captioning holds promise across various applications, including enhancing accessibility for visually impaired individuals, improving image search and retrieval based on the wearer's preferences, and augmenting user experience in diverse multimedia applications. Despite considerable progress, challenges persist, such as addressing long-term dependencies, enhancing diversity in generated captions, and developing models proficient in creating captions in multiple languages within the egocentric context.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction to the Project

1.1 Project Title

The title of the project is ”Egocentric Image Captioning”.

1.2 Project Description

Egocentric image captioning is a sophisticated project focused on developing a deep-learning model capable of generating coherent natural language descriptions for images captured from a first-person perspective (1). The primary objective is to train a neural network that can effectively articulate the details and context of an image in a manner resembling human-generated sentences. Similar to automatic image captioning, this undertaking involves intricate processes such as pre-processing the image data, formulating and training the neural network, and assessing the model’s efficacy through metrics like the BLEU score. The applications of egocentric image captioning are diverse, encompassing the creation of descriptive content for visually impaired individuals, augmenting search engine functionalities, and refining communication between humans and robots (2).

1.3 Scope of the Project

Egocentric image captioning projects encompass a wide scope with diverse applications. A key objective is to empower computers to comprehend the visual content from a first-person perspective and generate human-like natural language descriptions. The potential applications of egocentric image captioning are manifold, including aiding visually impaired individuals (3), enhancing the richness of online content with more immersive descriptions, and creating more intuitive interfaces for human-robot interaction. Additionally, this technology holds promise in sectors like e-commerce, advertising, and education. However, the development of accurate and dependable egocentric image captioning systems necessitates substantial advancements in the realms of computer vision, natural language processing, and deep learning. Hence, the scope of these projects is extensive and demands significant advancements to realize their full potential.

Chapter 2

2 Literature Review

2.1 Retrieval-based image captioning

Retrieval-based image captioning: Retrieval-based methods for captioning images were popular in earlier research. Given a query image, retrieval-based methods generate a caption for it by selecting one or more sentences from a pre-defined sentence pool. The generated caption may consist of a previously written statement or one that was put together using the recovered sentences (4). There are clear drawbacks to retrieval-based image captioning techniques. These techniques transmit properly constructed human-written sentences or phrases to create descriptions for image queries. Although the resulting outputs are frequently grammatically sound and fluid, limiting visual descriptions to preexisting words prevents them from adapting to unique object combinations or inventive scenarios. In other circumstances, created descriptions might even be irrelevant to the contents of the images.

2.2 Template-based image captioning

Another approach that is frequently used in early image captioning work is template based. Image captions are created using template-based methods using a syntactically and semantically limited process. Typically, a predetermined set of visual concepts must be detected to use a template-based method to generate a description for an image. Then, sentences are constructed using sentence templates, particular language grammar rules, or combinatorial optimization algorithms that connect the detected visual concepts (4). Syntactically proper sentences can be produced by template-based image captioning, and these descriptions are typically more accurate representations of the contents of the images than retrieval-based ones. The use of template-based approaches has drawbacks, though. There are restrictions on coverage, creativity, and complexity of generated sentences because description generation under the template-based framework is strictly limited to image contents recognized by visual models, with the typically small number of visual models available. Additionally, using rigid templates as the preceding sentence structures will make generated descriptions seem less natural compared to captions that humans wrote (5).

2.3 Reinforcement Learning

Reinforcement learning is a technique used in image captioning to train models to generate captions through a trial-and-error approach. It involves using an agent to learn how to maximize a reward signal based on the quality of the generated captions (6). In reinforcement learning-based image captioning, the agent interacts with an environment by generating captions and receiving feedback in the form of a reward signal. The agent learns to optimize the captioning process by trying different actions and observing the resulting reward signal. The reward signal can be based on metrics such as caption quality, diversity, and relevance. One approach to reinforcement learning-based image captioning is to use a framework called Policy Gradient. The Policy Gradient framework involves training an agent to learn a policy function that maps the input image to a distribution over possible captions. The agent then samples from this distribution to generate captions, and the quality of the generated captions is evaluated using a reward function. The agent learns to optimize the policy function by adjusting the weights of the network to maximize the expected reward. Another approach to reinforcement learning-based image captioning is to use a framework called Q-Learning (7). In this approach, the agent learns a Q-function that maps the input image and caption to a Q-value, which represents the expected reward of generating that caption for that image. The agent then selects the caption with the highest Q-value, and the quality of the generated captions is evaluated using a reward function. The agent learns to optimize the Q-function by adjusting the weights of the network to maximize the expected reward. Reinforcement learning-based image captioning has shown promising results in generating diverse and contextually relevant captions. However, it requires a large amount of training data and can be computationally expensive (7). Additionally, the quality of the generated captions depends heavily on the reward function used, which can be difficult to define and optimize.

2.4 Pre-trained Language Models

Pre-trained language models have been successfully used in a variety of natural language processing tasks, including image captioning. In this technique, a pre-trained language model is fine-tuned on a specific image captioning dataset (8). The idea is to leverage the pre-existing knowledge in the language model and adapt it to the specific task of image captioning. One popular pre-trained language model used in image captioning is the Bidirectional Encoder Representations from Transformers (BERT) (8). BERT is a transformer based architecture that uses a self-attention mechanism to learn contextual relations between words in a sentence. The pre-training is done on a large text corpus, and the resulting model can be fine-tuned for various downstream tasks, including image captioning. To use BERT for image captioning, the image features are first extracted using a pre-trained CNN. These features are then combined with the pre-trained BERT model to generate a textual description of the image. The CNN features are used as the initial input to the BERT model, and the model is fine-tuned on the image captioning dataset to generate a caption that describes the image. Another popular pre-trained language model used in image captioning is the Generative Pre-trained Transformer 2 (GPT-2) (8). GPT-2 is a transformer-based architecture that uses a similar self-attention mechanism as BERT but is trained in a generative manner. GPT-2 can be fine-tuned for various downstream tasks, including image captioning. In summary, pre-trained language models have been successfully used in image captioning by leveraging the pre-existing knowledge of the model and fine-tuning it on a specific image captioning dataset. This approach has shown promising results in generating more accurate and coherent image descriptions.

Chapter 3

3 Preliminaries

3.1 Encoder Decoder Architecture

Image captioning frameworks commonly use the Encoder-Decoder architecture (9), where the encoder part involves CNNs like ResNet-152, VGG-16, InceptionResNet, GoogleNet, etc., to extract features from input images. These features are then fed into the language model of the decoder to generate a sequence of words. The decoder typically includes recurrent networks like LSTM and GRU. The data is encoded into the desired format, such as transforming words from a spoken language into a one-dimensional vector known as the feature vector (10). Recurrent neural networks are stacked to create the encoder (11), which can comprehend the context and temporal relationships of sequences. The output of the encoder, the feature vector, represents the entire meaning of the input sequence in a one-dimensional vector. The length of the vector depends on the number of cells in the RNN. The decoder transforms the one-dimensional vector into the output sequence, which is the English phrase (11). It is constructed with RNN layers and a dense layer to predict the English term (10). One of the key features of this model is its ability to handle different input and output sequence lengths, enabling its use in applications such as question-and-answer sessions or video captioning. However, this straightforward encoder-decoder approach has a significant limitation of condensing all data into a one-dimensional vector (5), which can be challenging for lengthy input sequences. Recent advancements in image captioning research have focused on improving the encoder-decoder model by incorporating attention mechanisms, pre-trained language models (8), and reinforcement learning. These techniques have shown promising results in generating more accurate and natural language descriptions for images.

3.2 Word Embeddings

In image captioning, word embeddings play a crucial role in representing the semantics of words in a continuous vector space. Word embeddings capture the meaning of words by encoding them as dense and low-dimensional vectors, which are easier to handle and process than one-hot encoded vectors (12). The word embeddings are trained using large amounts of text data, such as news articles or books, and the knowledge captured from this training is then transferred to the image captioning task (13). During image captioning, the deep learning model maps the visual features of an image to a sequence of word embeddings, which are then decoded to generate a caption. The use of word embeddings allows the model to better capture the semantic relationships between words and generate more accurate and coherent captions. Furthermore, word embeddings can also help deal with the issue of out-of-vocabulary words, where the model has not seen a particular word before. By mapping the word to a vector in the embedding space, the model can still generate meaningful captions even for previously unseen words (12). Overall, word embeddings play a critical role in image captioning and have greatly improved the performance of deep learning models in generating accurate and natural language descriptions of images. Word embeddings are distributed representations of words in a high-dimensional space that captures the semantic meaning of words. There are several popular types of word embeddings, including fastText, GloVe, Word2Vec, and BERT (13).

- FastText: FastText (13) is a method developed by Facebook that uses characterlevel n-grams to represent words, enabling it to handle rare words and misspellings better than other methods.
- GloVe: GloVe (13) is a method that combines word co-occurrence statistics with a global matrix factorization technique to obtain embeddings that capture both local and global context.
- Word2Vec: Word2Vec (13) is a popular method that uses a neural network architecture to learn embedding based on the surrounding context of each word.

- BERT: BERT (12) which stands for Bidirectional Encoder Representations from Tran-
sformers, is a transformer-based model that learns contextualized embedding by considering the entire sentence or document.

Each of these types of word embedding has its strengths and weaknesses, and the choice of which one to use will depend on the specific needs of the task at hand.

3.3 Optimizers

An optimizer is a critical component of training neural networks, as it influences how the network learns and updates its parameters, such as weights and biases (14). The primary goal of an optimizer is to minimize the loss function by iteratively updating the network's parameters based on the gradients computed from the back propagation algorithm. There are several types of optimizers available, each with its own unique approach to minimizing the loss function (15). For instance, some optimizers use adaptive learning rates to adjust the step size of the updates, while others employ momentum to accelerate convergence. Choosing the right optimizer for a specific task is crucial as it can impact the training speed, convergence, and generalization performance of the model. Some popular optimizers used in image captioning include:

3.3.1 Adam

Adam (Adaptive Moment Estimation) optimizer combines the advantages of AdaGrad and RMSProp (14). It uses adaptive learning rates to update the model parameters, which helps to achieve faster convergence and better generalization (15). Adam is a popular optimizer used in image captioning because of its efficiency and robustness. It offers benefits that include:

- Easy to implement
- Low computer resource requirement
- Low Memory requirement

3.3.2 Adadelta

Adadelta optimizer uses a moving average of the squared gradients to adapt the learning rate for each parameter (14). Adadelta is particularly useful in scenarios where SGD has difficulty converging. The main advantage of AdaDelta is that we do not need to set a default learning rate but the drawback is that it requires more computational power (15).

3.3.3 AdaGrad

Adaptive Gradient Descent (AdaGrad) is an optimizer that adapts the learning rate for each parameter based on the historical gradient information (14). It performs well on sparse datasets but can struggle with non-convex optimization problems. The drawback is that if the neural network is deep the learning rate becomes very small number which will cause dead neuron problem (15). It offers benefits that include:

- With iterations, the learning rate changes adaptively
- It is well capable of training smaller data as well

3.3.4 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is the most basic optimizer that updates the model parameters in the direction of the negative gradient of the loss function (14). It works well for small datasets but can struggle with larger datasets due to its slow convergence (15). It offers benefits that include:

- Frequent updates of model parameter
- Requires less Memory
- Allows the use of large data sets as it has to update only one example at a time

3.3.5 RMSProp

Root Mean Square Propagation (RMSProp) is an optimizer that uses the moving average of the squared gradients to adapt the learning rate for each parameter (14). It helps to prevent the learning rate from becoming too large or too small and can be effective in optimizing deep neural networks. Here the learning rate gets adjusted automatically and it chooses a different learning rate for each parameter (15).

3.4 Activation Functions

3.4.1 RELU

ReLU (Rectified Linear Unit) is the most commonly used activation function in neural networks, including image captioning. ReLU sets all negative values to zero and keeps the positive values unchanged. This function is simple and fast, making it a popular choice for deep neural networks (16).

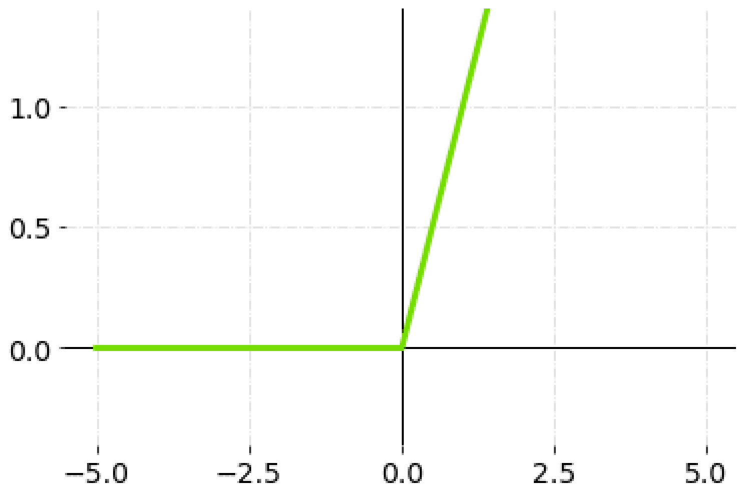


Figure 1: ReLU Activation Function (17)

3.4.2 Softmax

Softmax is used in the final layer of a neural network for multi-class classification problems. It transforms the output of the network into a probability distribution over the classes (16).

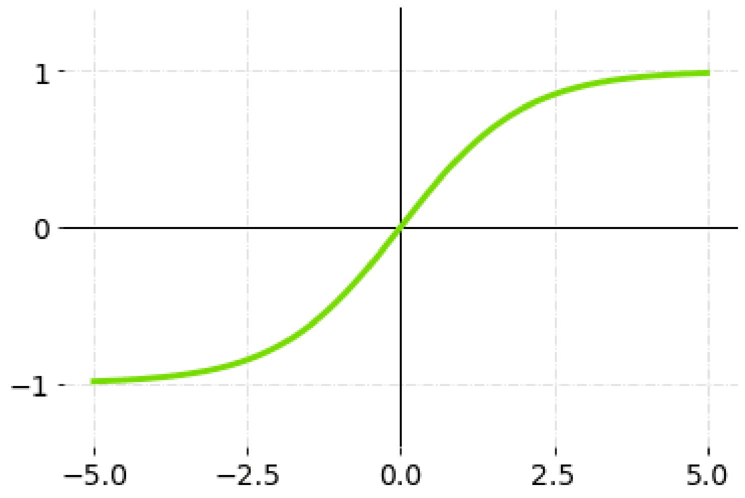


Figure 2: softmax Activation Function (17)

3.4.3 Tanh

Tanh (Hyperbolic Tangent) is also a commonly used activation function in image captioning. It squashes the input values between -1 and 1. Tanh is similar to the sigmoid function, but it has a steeper gradient, making it easier to train deep neural networks (16).

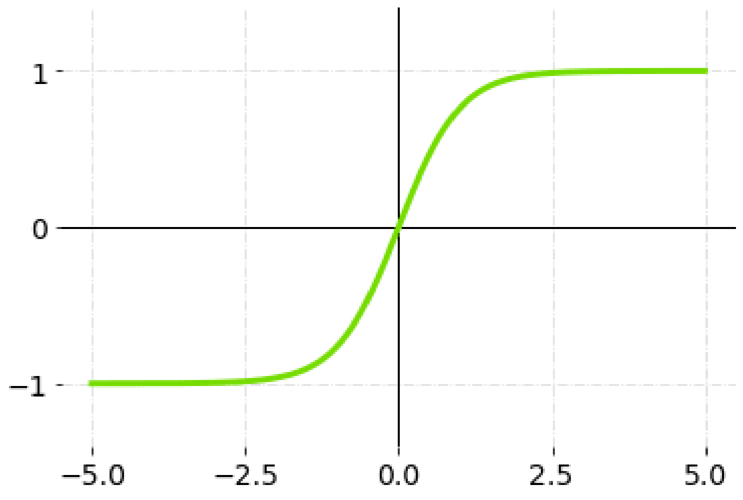


Figure 3: Tanh Activation Function (17)

3.4.4 Sigmoid

Sigmoid is another activation function used in image captioning. It maps the input values to a probability range between 0 and 1. Sigmoid is commonly used in binary classification problems (16).

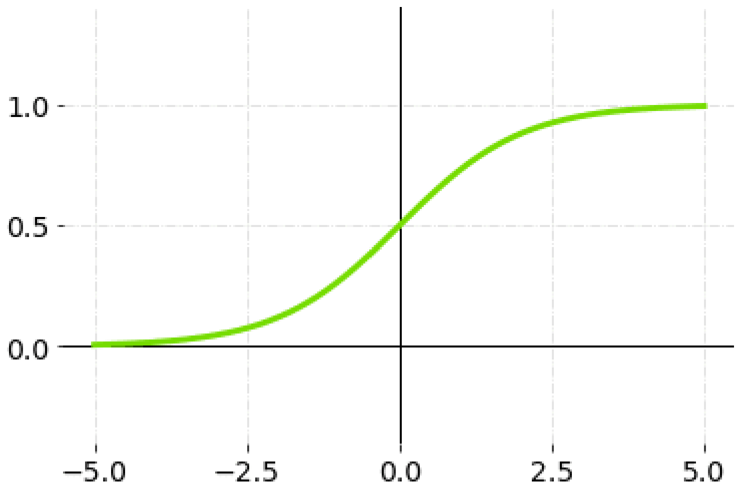


Figure 4: Softmax Activation Function (17)

3.4.5 Leaky RELU

Leaky ReLU is similar to ReLU, but it has a small slope for negative input values. This helps to prevent the dying ReLU problem, where a neuron becomes permanently inactive (16).

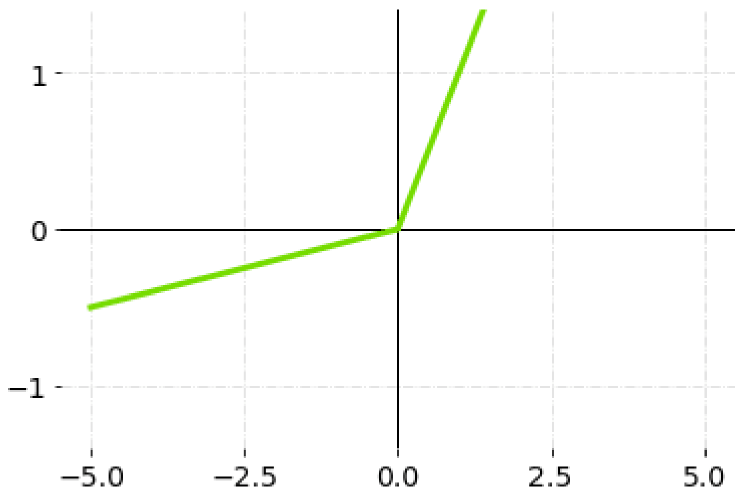


Figure 5: Leaky RELU Activation Function (17)

3.5 Metrics

3.5.1 Accuracy

Accuracy is the percentage of correct predictions made by the model on the training dataset (14). In image captioning, accuracy is a metric used to evaluate how well the model can generate captions that match the actual captions (5).

3.5.2 Loss

Loss is the difference between the predicted output of the model and the actual output for a given input (14). The objective of training a model is to minimize this difference (5). In image captioning, loss is used to evaluate how well the model is able to predict the correct captions.

3.5.3 Validation Accuracy

Validation accuracy is the percentage of correct predictions made by the model on a validation dataset (14). The purpose of using a validation dataset is to evaluate the performance of the model on new, unseen data (5). In image captioning, validation accuracy is used to evaluate how well the model can generalize to new images and generate captions that match the actual captions.

3.5.4 Validation Loss

Validation loss is the difference between the predicted output of the model and the actual output for a given input in the validation dataset (14). The purpose of using a validation dataset is to evaluate the performance of the model on new, unseen data (5). In image captioning, validation loss is used to evaluate how well the model is able to predict the correct captions for new, unseen images.

3.6 Evaluation Metrics

Evaluation metrics offer numerical measurements to rate the effectiveness of captioning models. The quality and relevance of generated captions cannot be fully captured by a single metric; therefore, numerous metrics are accounted for this task. The evaluation measures to be utilized will depend on the specific problem being handled and the anticipated outcomes.

3.6.1 BLEU

A popular metric for assessing the quality of machine-generated text, such as machine translation or text generation, is called BLEU (Bilingual Evaluation Understudy). It gauges how closely the generated text resembles a reference text that was written by a human. Higher scores denote better quality and closer relation to the reference text, and the BLEU derives a value between 0 and 1. Comparing the n-grams (contiguous word sequences) in the generated text with those in the reference text is the fundamental concept underpinning BLEU (18). How many n-grams in the generated text match those in the reference text is determined by the n-gram precision. Additionally, BLEU favors conciseness, so shorter n-gram matches score higher (19).

3.6.2 METEOR

METEOR (Metric for Evaluation of Translation with Explicit Ordering) is an extensive measurement metric that assesses the effectiveness of generated text by taking into account both recall and precision. To provide a more complex analysis, it also adds other elements including word order, synonymy, and stemming (20). METEOR seeks to quantify the degree of correspondence between the generated text and the reference text.METEOR uses a variety of ways to improve its accuracy when comparing the generated text with the reference text. By reducing words to their root form, stemming, for instance, makes it possible to match similar keywords more effectively. In order to guarantee that the metric detects semantically equivalent statements, synonymy takes into account distinct terms that convey similar meanings. Word order consideration also aids in assessing the coherence and fluency of the output text (21).

3.6.3 CIDEr

CIDEr (Consensus-based Image Description Evaluation) is a specialized evaluation metric developed specifically for picture captioning tasks. It aims to assess the quality of generated captions by comparing them to a set of reference captions, capturing the level of agreement and consensus among the references. To compute the CIDEr score, Term Frequency-Inverse Document Frequency (TF-IDF) weighted word frequencies are utilized. TF-IDF is a numerical representation that reflects the importance of words within a document relative to their occurrence in a larger corpus. By incorporating TF-IDF, CIDEr gives more weight to informative words that are less common. The core computation of CIDEr involves calculating the cosine similarity between the TF-IDF weighted word frequencies of the generated and reference captions. Cosine similarity measures the similarity between two vectors by examining the cosine of the angle between them. In the case of CIDEr, the vectors represent the TF-IDF weighted word frequencies of the captions (22).

3.6.4 SPICE

An emerging metric in the realm of image caption evaluation is SPICE, which stands for Semantic Propositional Image Caption Evaluation. Unlike traditional metrics, SPICE delves deeper into assessing image caption similarity by examining the agreement between the candidate sentence’s scene graph tuples (23) and all reference sentences. The introduction of SPICE to the evaluation toolkit signifies a shift towards more nuanced and semantically rich assessments. By considering the structural alignment between the candidate caption and the scene graph representations, SPICE offers a more sophisticated perspective on caption quality

3.6.5 ROUGE

ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, represents a vital arsenal of evaluation measures that are conventionally associated with text summarization tasks. However, the utility of ROUGE extends seamlessly into the realm of image captioning. Leveraging n-gram matching as its cornerstone technique, ROUGE quantifies the degree of textual overlap between the generated image captions and their reference counterparts (24). By focusing on recall-oriented ratings, ROUGE assesses how effectively the generated captions encapsulate the informational essence present in the reference captions.

3.6.6 SPIDEr

SPIDEr is a relatively new evaluation metric that combines SPICE and CIDEr. It was developed for more effective searching of qualitative research.

3.6.7 WMD

The word mover's distance (WMD) is a method for calculating how similar two papers are to one another. The smallest distance that the embedded words in one text document must ”travel” in order to reach the embedded words in the other text document is used to compare the dissimilarity between the two texts. WMD strives for text document similarity by focusing on both semantic and syntactic approaches. By using an optimal transport formulation, it can benefit from the word space's underlying shape (25)

Chapter 4

4 Methodology

4.1 Project Approach

Understanding the task and requirements, gathering, and preprocessing captured image dataset of egocentric images, designing a transformer based architecture , implementing and training the model using deep learning, assessing performance using metrics and qualitative analysis, fine-tuning the model based on results, and deploying the model for production or integration are the steps involved in creating an egocentric image captioning project. Throughout the project, good documentation, sound coding techniques, and getting feedback are crucial.

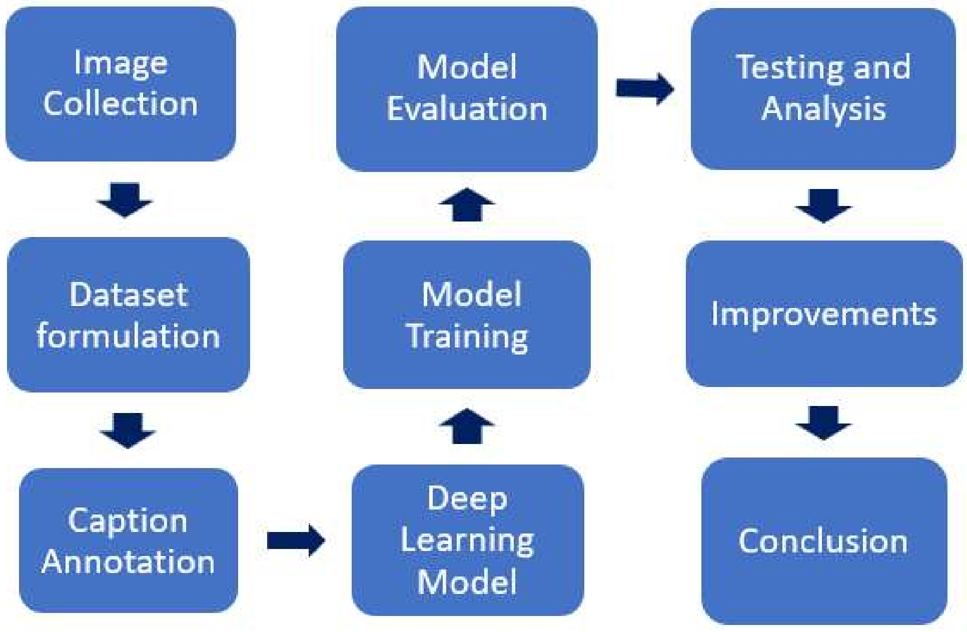


Figure 6: Methodology

4.2 Step 1: Dataset Creation

The dataset presented here comprises 20 distinct classes captured through an ego-centric perspective, offering a diverse collection of everyday objects and entities. The dataset creation process involved meticulous curation and annotation to ensure the quality and diversity of the data. Each class, ranging from common items like Bottles, Dogs, and Chairs to more specific entities like Wall Clocks and Laptops, was systematically compiled to simulate a first-person visual experience. The inclusion of various objects aims to provide a comprehensive representation of our surroundings. This dataset serves as a valuable resource for training and evaluating computer vision models across a spectrum of applications. Its systematic compilation, coupled with careful annotation, establishes a robust foundation for advancing machine learning algorithms in recognizing and understanding visual information from an ego-centric viewpoint.

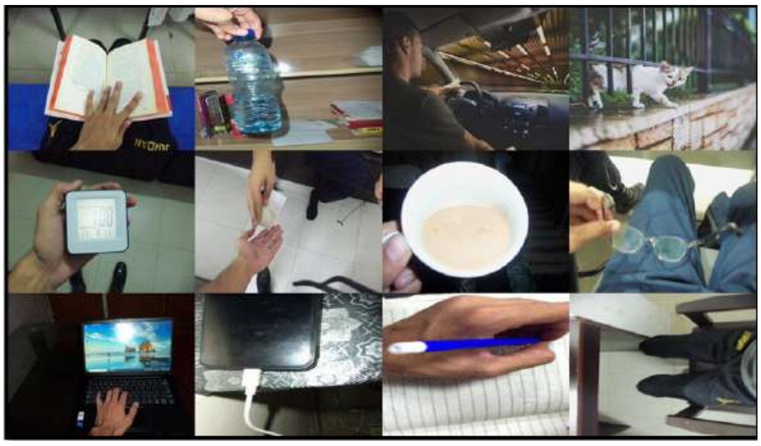


Figure 7: Egocentric Images Dataset

Dataset	Training Split	Validation Split	Testing Split	Total Images
Self Made	400 per class	50 per class	50 per class	500 per class

Table 1: Dataset Split

4.3 Step 2: Caption Annotation

The dataset was meticulously annotated with three distinct captions assigned to each individual image. This captioning process was undertaken to enrich the dataset with comprehensive textual descriptions that encapsulate diverse perspectives, context, and details associated with the visual content of the images. The inclusion of multiple captions per image serves to enhance the dataset's depth, providing researchers and practitioners with a multifaceted resource for training and evaluating algorithms in image understanding and related applications. The triple-captioning approach not only contributes to a nuanced representation of the dataset but also facilitates the development and assessment of models capable of interpreting and generating varied textual descriptions for a given image. This formal annotation strategy adheres to best practices in dataset creation, promoting versatility and adaptability for a wide range of research endeavors and machine learning applications(26).

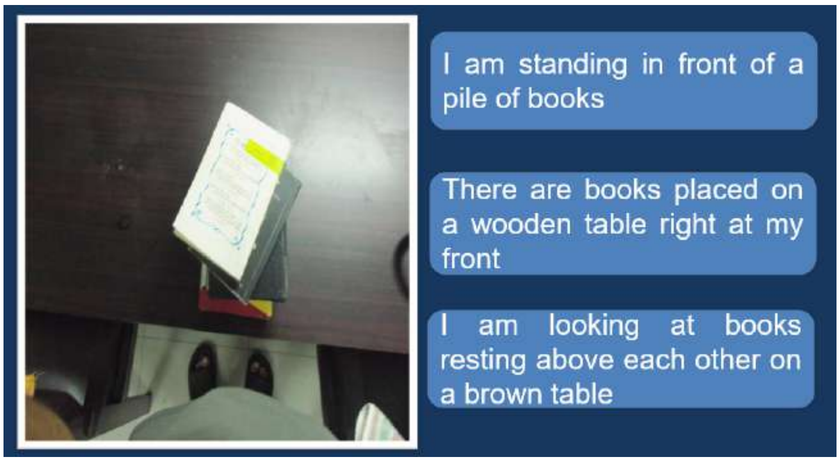

I am standing in front of a pile of books
There are books placed on a wooden table right at my front
I am looking at books resting above each other on a brown table

Figure 8: Captions Annotations

4.4 Step 3: Image Pre-processing

In image captioning, before feeding the images to the model for training, several pre-processing steps are required to be performed to ensure better performance and faster convergence of the model. One of the crucial steps is image pre-processing, which involves a series of transformations to make the input images suitable for processing by the model. Image pre-processing typically includes steps like resizing, cropping, normalization, and array conversion. These are explained below:

- Resizing is done to ensure that all images in the dataset have the same dimensions (27), which is a prerequisite for feeding them to the model.
- Normalization is the process of scaling pixel values to a specific range (27), typically [0,1] or [-1,1].
- Array conversion is used to convert the image into a suitable numerical representation for processing by the model. Proper image pre-processing is crucial for the success

of the image captioning model, as it can significantly impact the accuracy and speed of convergence of the model (27).

4.5 Step 4: Text Pre-processing

Text pre-processing is a crucial step in natural language processing tasks like image captioning. The goal is to clean and normalize the text data to make it easier for the model to learn meaningful patterns. Text cleaning typically involves lower-casing all characters and removing any unwanted characters like punctuation's, numbers, and special characters (28). Tokenization is another important step in text pre-processing, where the text is broken down into individual words or tokens. This process allows the model to understand the structure of the text and learn the relationships between words (29).

4.6 Step 5: Choose Word Embedding

BERT (Bidirectional Encoder Representations from Transformers) embeddings have emerged as a state-of-the-art solution for natural language processing tasks, including image captioning. Unlike traditional word embeddings, BERT embeddings are contextualized, taking into account the entire context of a word within a sentence. This bidirectional modeling allows BERT to capture intricate relationships and nuances in language, leading to highly informative representations. With a pre-training phase on massive text corpora and a focus on subword tokenization, BERT embeddings excel in handling out-of-vocabulary words and capturing both semantic and syntactic information(30). The ability to transfer knowledge from pre-training to specific downstream tasks makes BERT embeddings a versatile and powerful tool in image captioning, enabling models to understand context, generate coherent captions, and achieve state-of-the-art performance(31).

4.7 Step 6: Multimodal Model

To pre-train a unified vision-language model with both understanding and generation capabilities, BLIP introduces a multimodal mixture of an encoder-decoder and a multitask model which can operate in one of the three modes:

- Unimodal encoders, which separately encode images and text. The image encoder is a vision transformer. The text encoder is the same as BERT.
- Image-grounded text encoder, which injects visual information by inserting a cross-attention layer between the self-attention layer and the feed-forward network for each transformer block of the text encoder.
- Image-grounded text decoder, which replaces the bi-directional self-attention layers in the text encoder with causal self-attention layers.

BLIP jointly optimizes three objectives during pre-training, with two understanding-based objectives (ITC, ITM) and one generation-based objective (LM):

- Image-Text Contrastive Loss (ITC) activates the unimodal encoder. It aims to align the feature space of the visual transformer and the text transformer by encouraging positive image-text pairs to have similar representations in contrast to the negative pairs.
- Image-Text Matching Loss (ITM) activates the image-grounded text encoder. ITM is a binary classification task, where the model is asked to predict whether an image-text pair is positive (matched) or negative (unmatched) given their multimodal feature.
- Language Modeling Loss (LM) activates the image-grounded text decoder, which aims to generate textual descriptions conditioned on the images.

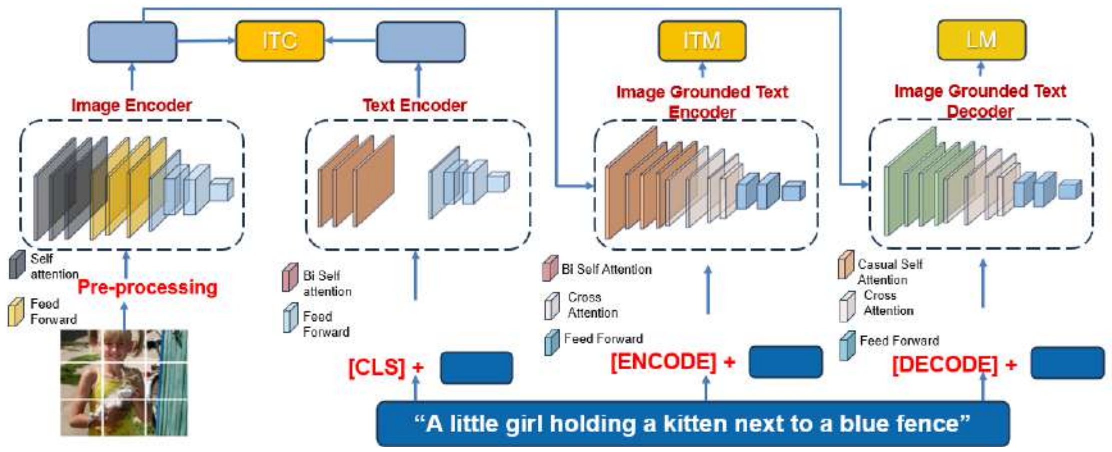


Figure 9: Model Architecture

4.8 Step 7: Training the Model

The final step is to train the model. During model training, it is essential to monitor the validation loss to prevent overfitting and ensure that the model is generalizing well to unseen data. The training process typically involves iterating through the dataset multiple times, or epochs, adjusting the model's weights and biases after each iteration to minimize the loss function. However, training the model for too long can lead to overfitting, where the model becomes too specialized to the training data and performs poorly on new data. To prevent overfitting, the model is trained until the validation loss stops decreasing and starts increasing. At this point, the model has learned all the meaningful patterns in the data and further training will only lead to overfitting. By stopping the training at this point, we ensure that the model has the best possible performance on new data.

Chapter 5

5 Training Results

Initially the models training accuracy and loss curve showed a liner increase and decrease but exactly after two epochs, the model started overfitting on the dataset so two epochs are set as benchmark to stop the model training and the weights are saved.

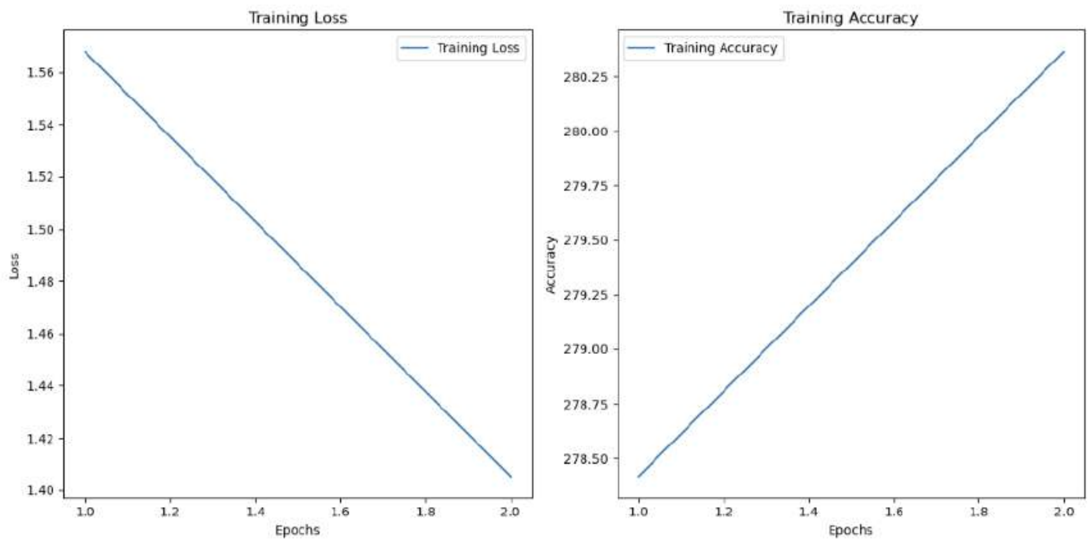


Figure 10: Loss and Accuracy Curve first epoch

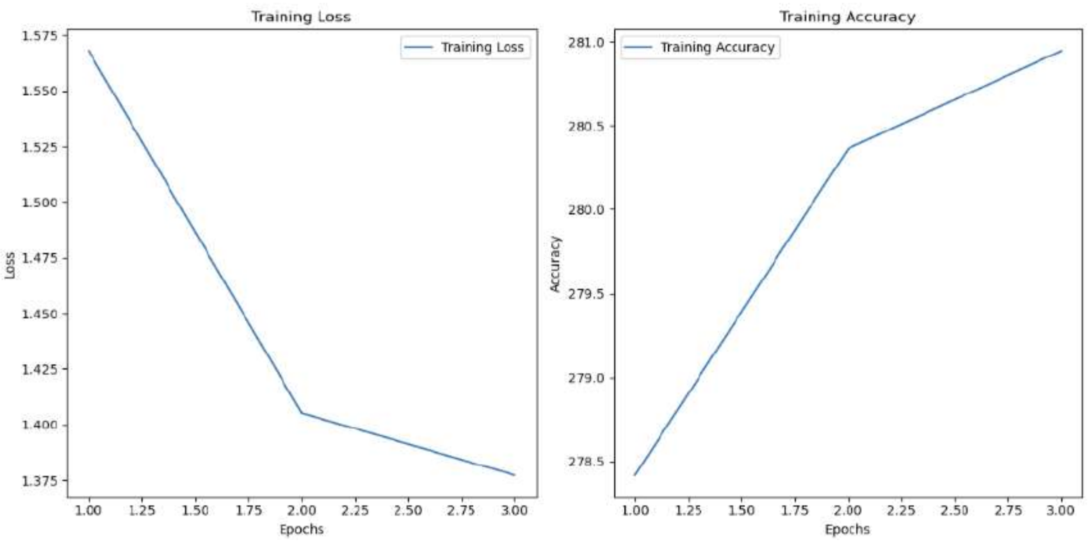
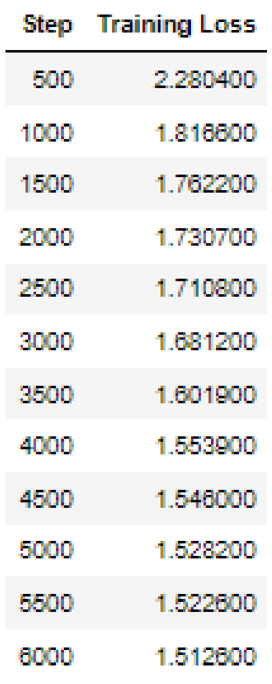
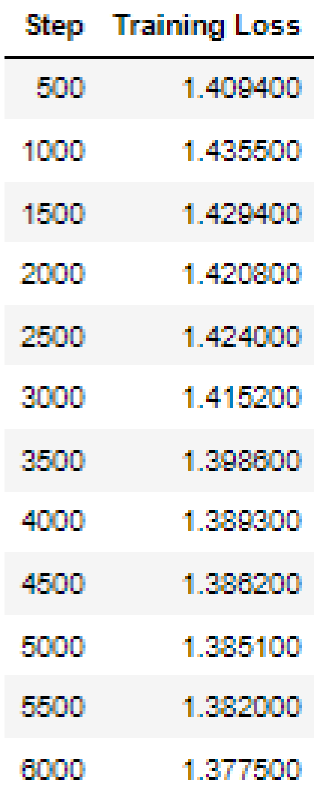


Figure 11: Loss and Accuracy Curve second epoch



Step	Training Loss
500	2.280400
1000	1.816600
1500	1.762200
2000	1.730700
2500	1.710800
3000	1.681200
3500	1.601900
4000	1.553900
4500	1.546000
5000	1.528200
5500	1.522600
6000	1.512600

Figure 12: Training Loss



Step	Training Loss
500	1.409400
1000	1.435500
1500	1.429400
2000	1.420800
2500	1.424000
3000	1.415200
3500	1.398600
4000	1.389300
4500	1.386200
5000	1.385100
5500	1.382000
6000	1.377500

Figure 13: Training Loss

Chapter 6

6 Quantitative Results

Metric	Value
BLEU-1	0.6853
BLEU-2	0.5877
BLEU-3	0.5227
BLEU-4	0.47765
ROUGE-1	0.4999
ROUGE-2	0.3520
ROUGE-L	0.4779
METEOR	0.4979

The values of different parameters used to get the scores are:

- `image_text_hidden_size` = 256
- `initializer_factor` = 1.0
- `initializer_range` = 0.02
- `logit_scale_init_value` = 2.6592
- `model_type` = blip
- `projection_dim` = 512

text_config

- `encoder_hidden_size` = 1024
- `initializer_factor` = 1.0
- `model_type` = blip_text_model
- `num_attention_heads` = 12

vision_config

- `dropout` = 0.0
- `hidden_size` = 1024

- `initializer_factor` = 1.0
- `initializer_range` = 0.02
- `intermediate_size` = 4096
- `model_type` = blip_vision_model
- `num_attention_heads` = 16
- `num_channels` = 3
- `num_hidden_layers` = 24

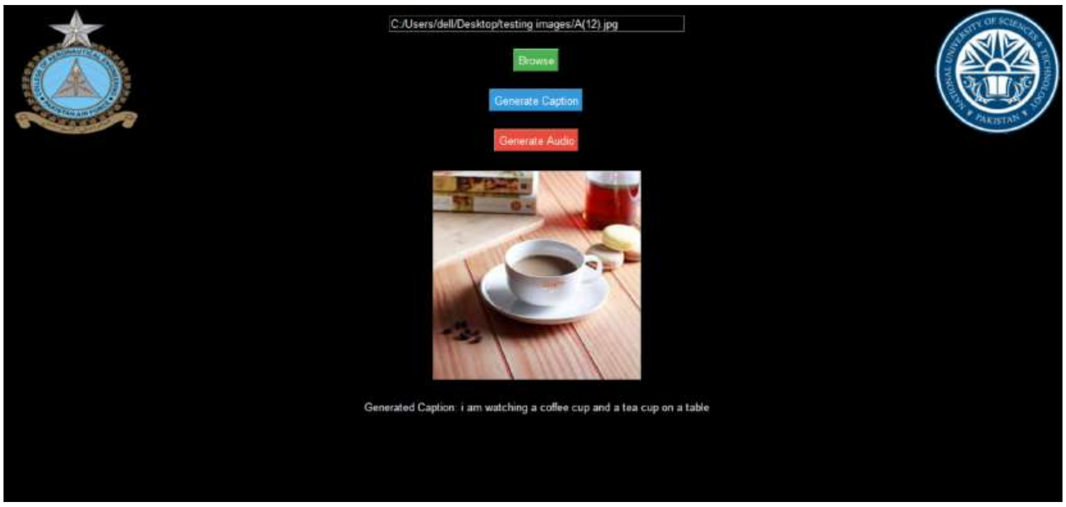
Training Parameters

- `num_train_epochs` = 2
- `learning rate` = 1e-5
- `per_device_train_batch_size` = 2
- `save_steps` = 500
- `save_total_limit` = 20

Chapter 7

7 Qualitative Results

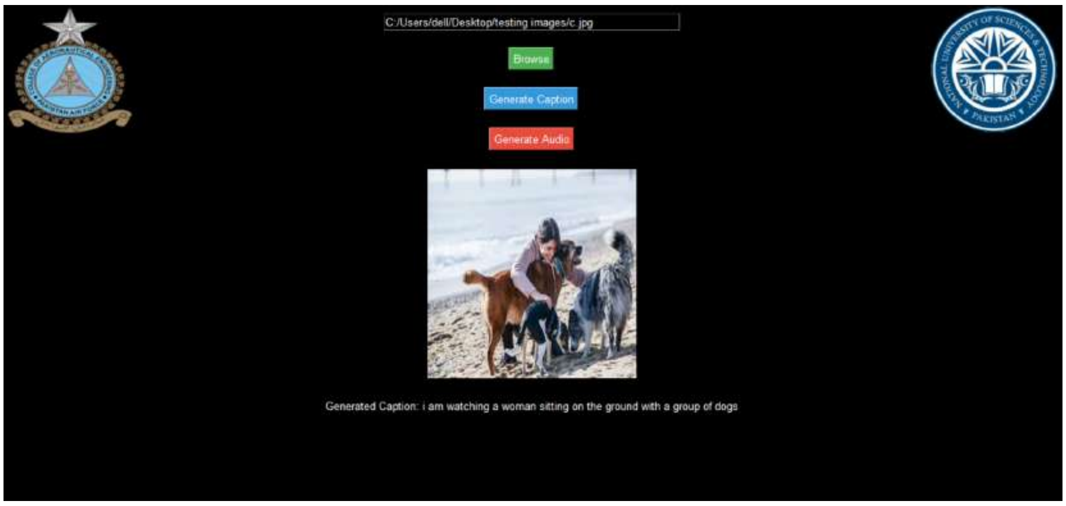
Below are the captions generated by the model on 6 different random images.



C:/Users/dell/Desktop/testing images/A(12).jpg

Generated Caption: i am watching a coffee cup and a tea cup on a table

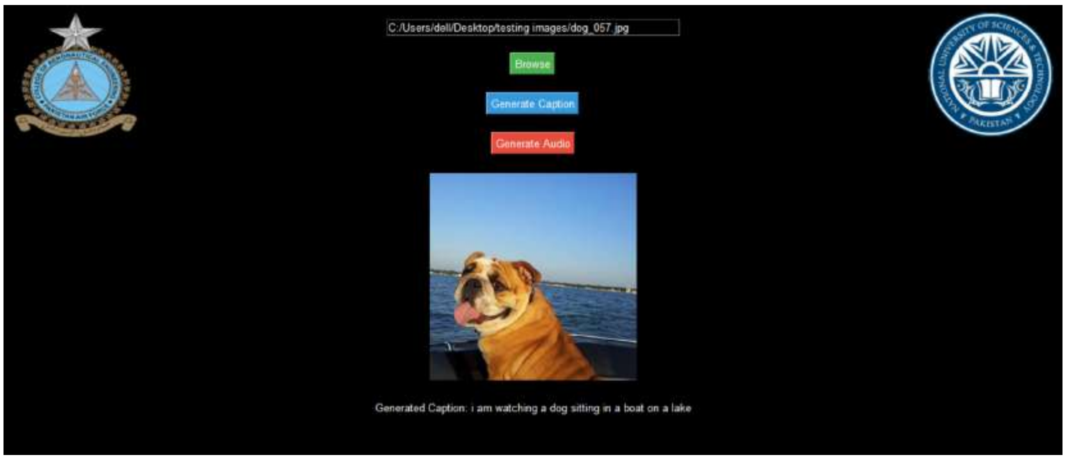
Figure 14: Caption Generated for First Image



C:/Users/dell/Desktop/testing images/c.jpg

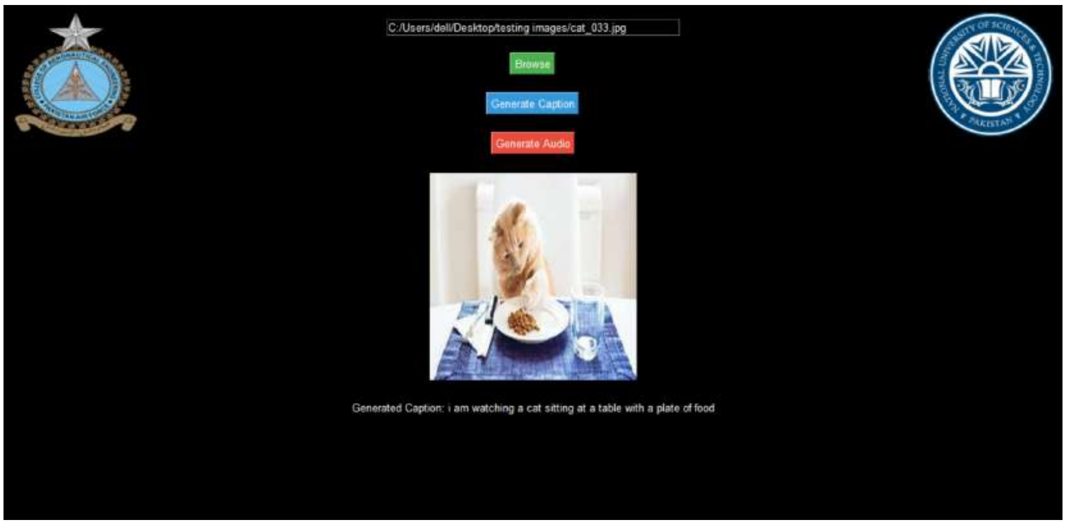
Generated Caption: i am watching a woman sitting on the ground with a group of dogs

Figure 15: Caption Generated for Second Image



Generated Caption: i am watching a dog sitting in a boat on a lake

Figure 16: Caption Generated for Fourth Image



Generated Caption: i am watching a cat sitting at a table with a plate of food

Figure 17: Caption Generated for Fifth Image

Chapter 8

8 Challenges

Numerous challenges and problems were encountered during this project which include:

- High computational power requirement: Image captioning is a computationally intensive task that requires large amounts of computing resources (32). Training deep learning models on large datasets like MS COCO can take several days or even weeks on a single machine, which can be a bottleneck for many researchers and developers.
- High VRAM graphics card requirement: Training deep learning models on large datasets requires a lot of memory, and many models cannot fit in the memory of a typical graphics card. This can limit the size of the models that can be trained and can also result in slower training times due to the need to swap data in and out of memory (32).
- Overfitting: One of the biggest challenges in training deep learning models for image captioning is overfitting. This occurs when a model becomes too specialized to the training data and is unable to generalize well to new data. Overfitting can be mitigated by using techniques like early stopping, regularization, and dropout.
- Parameter tuning: Determining the best values for parameters like the number of layers, number of hidden units, learning rate, and choice of optimizer can be a challenge. There are many different approaches to hyperparameter tuning, including manual tuning, grid search, and random search.
- RAM limitations: Training deep learning models on large datasets can require a lot of memory, and many machines do not have enough RAM to load the entire dataset into memory at once (32). This can be addressed using techniques like data generators, which load data in batches during training, but this can also result in slower training times due to the need to read data from a disk.

Chapter 9

9 Applications

Image captioning has various applications in different fields. Some of them are:

- Tactical Awareness:In military or law enforcement scenarios, egocentric image captioning can enhance tactical awareness by providing real-time, contextual information to personnel. Captions can describe the surroundings, identify potential threats, and offer valuable insights during missions.
- Intelligence Gathering: Egocentric image captioning can be utilized for intelligence gathering purposes. Captions can describe the details of a scene, such as people, objects, and activities, aiding in the analysis of captured visual data for security or investigative purposes.
- Visually Impaired: Egocentric image captioning has applications in assisting visually impaired individuals. Wearers of egocentric cameras can receive real-time descriptions of their surroundings, improving spatial awareness and facilitating independent navigation.
- Aircraft Maintenance: Maintenance personnel wearing egocentric cameras can use image captions to document and communicate issues related to aircraft maintenance. This assists in recording the condition of components, identifying faults, and facilitating efficient collaboration among maintenance teams.
- Evidence Collection: In legal and forensic contexts, egocentric image captions can serve as valuable evidence by providing detailed descriptions of the scene. This can be crucial for documenting incidents, aiding investigations, and presenting visual information in a courtroom setting.
- Surveillance: Egocentric image captioning enhances surveillance applications by automatically describing the observed environment. It enables security personnel to

monitor and understand activities in real-time, making it easier to identify potential security threats or unusual behavior.

- Reconnaissance: Military or exploration personnel can benefit from egocentric image captioning during reconnaissance missions. Captions can describe terrain, landmarks, and potential obstacles, aiding in decision-making and mission planning.
- Education: Egocentric image captioning can be used in educational settings to enhance learning experiences. It allows students or instructors to capture and describe practical demonstrations, experiments, or field trips, providing additional context to educational content.
- Medical: In healthcare, egocentric image captioning can assist medical professionals during surgeries or examinations. It enables the documentation of procedures, patient conditions, and equipment usage, supporting training and enhancing patient care.
- Social Media Advertisement: Individuals or businesses can use egocentric image captioning for creating engaging content on social media. Captions can provide context to personal experiences, product demonstrations, or travel adventures, making content more relatable and appealing to viewers.

Chapter 10

10 Conclusion and Recommendations

10.1 Conclusion

In conclusion, image captioning is a challenging and exciting field of research that has seen significant progress in recent years. With the availability of large datasets, powerful deep learning models, and advanced natural language processing techniques, image captioning has become a popular area of research for both academics and industry practitioners. Image captioning has many potential applications, including helping visually impaired individuals to understand images, improving image search, and enhancing the user experience in social media and e-commerce platforms. Although there are still many challenges to overcome, the continued development of image captioning technology is likely to have a significant impact on many fields and change the way we interact with visual media.

10.2 Recommendations

To enhance the performance of egocentric image captioning models, it is crucial to leverage datasets specifically designed for first-person perspectives, such as the EPIC-Kitchens or EGTEA Gaze+ datasets, which provide diverse scenes and activities relevant to egocentric scenarios. Annotating these datasets with descriptive captions that encapsulate not only visible objects but also the wearer's interactions and intentions is essential. Fine-tuning pre-trained models, particularly those based on transformer architectures, on egocentric datasets facilitates adaptation to the unique challenges of egocentric image captioning. Incorporating attention mechanisms into the model architecture enhances its ability to focus on relevant features in egocentric images, ensuring more accurate and contextually rich captions from the wearer's point of view. Additionally, emphasizing contextual understanding, user interaction modeling, and exploring multi-modal fusion with other sensory inputs can further improve the model's descriptive capabilities. Evaluation in real-world scenarios, continuous learning, and addressing ethical considerations, especially concerning user privacy, contribute to the overall effectiveness and ethical deployment of egocentric image captioning applications.

Appendices

A Program Code

A.1 Code for Scenario 01: Image and Text Pre-processing

```python

import matplotlib
from PIL import Image
from transformers import BlipImageProcessor
import os  #to interact with operating system
import pandas as pd   # to manupulate data and offers data structure
from PIL import Image  # for opening saving and minupulation of images
from transformers import BlipProcessor, BlipForConditionalGeneration, Trainer,
↪ TrainingArguments
from torch.utils.data import Dataset
from torchvision import transforms
import torch

class CustomBatchEncoding:
    def __init__(self, data_dict):
        self._dict_ = data_dict

class ImageCaptioningDataset(Dataset):
    def __init__(self, image_folder, csv_file, processor, max_length=32):
        self.image_folder = image_folder
        self.df = pd.read_csv(csv_file)
        self.processor = processor
        self.max_length = max_length

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        img_filename = os.path.join(self.image_folder, self.df.iloc[idx, 0])
        image = Image.open(img_filename).convert("RGB")

        # Choose one of the captions randomly (you can modify this based on your
        ↪ preference)
        caption = self.df.iloc[idx, 1]

        # Preprocess the image and caption using the provided processor
        inputs = self.processor(image, caption, return_tensors="pt",
        ↪  padding="max_length", max_length=self.max_length, truncation=True)

        # Remove batch dimension
        inputs = {k: v.squeeze() for k, v in inputs.items()}

        # Add labels for training
        inputs["labels"] = inputs["input_ids"].clone()

        # Convert all elements to the same data type (torch.Tensor)
        inputs = {k: torch.tensor(v) for k, v in inputs.items()}

        return inputs

def setup_paths():
    # Set paths for your dataset
    image_folder_path = "C:/Users/DELL/Desktop/books"
    csv_file_path = "C:/Users/DELL/Desktop/image_captions_books.csv"

    if not os.path.exists(image_folder_path) or not os.path.exists(csv_file_path):
            raise FileNotFoundError("Image folder or CSV file not found.")

    return image_folder_path, csv_file_path

# Example usage:
image_folder_path, csv_file_path = setup_paths()
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-large")

# Create an instance of the custom dataset
custom_dataset = ImageCaptioningDataset(image_folder_path, csv_file_path, processor)

# Example: Print the first item in the dataset
print(custom_dataset[0])

# Define training arguments
training_args = TrainingArguments(
    output_dir="C:/Users/DELL/Desktop/weight",
    num_train_epochs=3,
    per_device_train_batch_size=2,
    save_steps=1000,
    save_total_limit=2,
)

# Instantiate the BlipImageProcessor
processor = BlipImageProcessor(do_resize=True, size={"height": 384, "width": 384},
↪   do_rescale=True, do_normalize=True)


# Load an image from your folder
image_path = "C:/Users/DELL/Desktop/abc.jpg"
image = Image.open(image_path)

# Preprocess the image
processed_image = processor.preprocess(image, return_tensors="np")

# Access the preprocessed pixel values
pixel_values = processed_image["pixel_values"]
print(pixel_values)
print(processor)

```

A.2 Code for Scenario 02: Image and Text Mapping

```python

import os
import datasets
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from PIL import Image
from pathlib import Path
from tqdm.auto import tqdm
import multiprocessing as mp
from sklearn.model_selection import train_test_split

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import io, transforms
from torch.utils.data import Dataset, DataLoader, random_split

from sklearn.model_selection import train_test_split
import pandas as pd

def load_and_split_data(file_path, test_size=0.8):
    df = pd.read_csv(file_path)
    train_df, val_df = train_test_split(df, test_size=test_size)
    return train_df, val_df, df

train_df, val_df, df = load_and_split_data("C:/Users/DELL/Desktop/DATASET
↪  FINAL/image_captions (books).csv")
df.head(10)

import matplotlib.image as mpimg

captions_df = pd.read_csv(r'C:/Users/DELL/Desktop/DATASET FINAL/image_captions
↪  (books).csv', delimiter='\t', header=None, names=['image', 'caption'])

```

```
image_dir = "C:/Users/DELL/Desktop/DATASET FINAL/book"


# Number of images to display
num_images = 5

# Randomly sample images
sampled_images = captions_df.sample(num_images)

# Plot images and their captions
fig, axes = plt.subplots(nrows=num_images, figsize=(5, 5*num_images))

for ax, (img_name, caption) in zip(axes, sampled_images.values):
    img_path = os.path.join(image_dir, img_name.split(',')[0])
    caption = img_name.split(',')[1]
    img = mpimg.imread(img_path)
    ax.imshow(img)
    ax.set_title(caption)
    ax.axis('on')

plt.tight_layout()
plt.show()
```

A.3 Code for Scenario 03: Model Architecture

```python


import transformers
from models.med import BertConfig, BertModel, BertLMHeadModel
from transformers import BertTokenizer
import transformers
transformers.logging.set_verbosity_error()

import torch
from torch import nn
import torch.nn.functional as F

from models.blip import create_vit, init_tokenizer, load_checkpoint

class BLIP_Pretrain(nn.Module):
    def __init__(self,
                 med_config = 'configs/bert_config.json',
                 image_size = 224,
                 vit = 'base',
                 vit_grad_ckpt = False,
                 vit_ckpt_layer = 0,
                 embed_dim = 256,
                 queue_size = 57600,
                 momentum = 0.995,
                 ):
        """
        Args:
            med_config (str): path for the mixture of encoder-decoder model's
            ↪ configuration file
            image_size (int): input image size
            vit (str): model size of vision transformer
        """
        super().__init__()

        self.visual_encoder, vision_width = create_vit(vit,image_size, vit_grad_ckpt,
        ↪ vit_ckpt_layer, 0)

        if vit=='base':
            checkpoint = torch.hub.load_state_dict_from_url(

                ↪  url="https://dl.fbaipublicfiles.com/deit/deit_base_patch16_224-b5f2ef4d.pth",
                map_location="cpu", check_hash=True)
            state_dict = checkpoint["model"]
            msg = self.visual_encoder.load_state_dict(state_dict,strict=False)
        elif vit=='large':
            from timm.models.helpers import load_custom_pretrained
            from timm.models.vision_transformer import default_cfgs

            ↪  load_custom_pretrained(self.visual_encoder,default_cfgs['vit_large_patch16_224_in21k

        self.tokenizer = init_tokenizer()
        encoder_config = BertConfig.from_json_file(med_config)
        encoder_config.encoder_width = vision_width
        self.text_encoder =
        ↪  BertModel.from_pretrained('bert-base-uncased',config=encoder_config,
        ↪  add_pooling_layer=False)
        self.text_encoder.resize_token_embeddings(len(self.tokenizer))

        text_width = self.text_encoder.config.hidden_size

        self.vision_proj = nn.Linear(vision_width, embed_dim)
        self.text_proj = nn.Linear(text_width, embed_dim)

        self.itm_head = nn.Linear(text_width, 2)

        # create momentum encoders
        self.visual_encoder_m, vision_width = create_vit(vit,image_size)
        self.vision_proj_m = nn.Linear(vision_width, embed_dim)
        self.text_encoder_m = BertModel(config=encoder_config, add_pooling_layer=False)
        self.text_proj_m = nn.Linear(text_width, embed_dim)

        self.model_pairs = [[self.visual_encoder,self.visual_encoder_m],
                            [self.vision_proj,self.vision_proj_m],
                            [self.text_encoder,self.text_encoder_m],
                            [self.text_proj,self.text_proj_m],
                           ]
        self.copy_params()

        # create the queue
        self.register_buffer("image_queue", torch.randn(embed_dim, queue_size))
        self.register_buffer("text_queue", torch.randn(embed_dim, queue_size))
        self.register_buffer("queue_ptr", torch.zeros(1, dtype=torch.long))

        self.image_queue = nn.functional.normalize(self.image_queue, dim=0)
        self.text_queue = nn.functional.normalize(self.text_queue, dim=0)

        self.queue_size = queue_size
        self.momentum = momentum
        self.temp = nn.Parameter(0.07*torch.ones([]))

        # create the decoder
        decoder_config = BertConfig.from_json_file(med_config)
        decoder_config.encoder_width = vision_width
        self.text_decoder =
         ↪  BertLMHeadModel.from_pretrained('bert-base-uncased',config=decoder_config)
        self.text_decoder.resize_token_embeddings(len(self.tokenizer))

         ↪  tie_encoder_decoder_weights(self.text_encoder,self.text_decoder.bert,'','/attention')


    def forward(self, image, caption, alpha):
        with torch.no_grad():
            self.temp.clamp_(0.001,0.5)

        image_embeds = self.visual_encoder(image)
        image_atts =
         ↪  torch.ones(image_embeds.size()[:-1],dtype=torch.long).to(image.device)
        image_feat = F.normalize(self.vision_proj(image_embeds[:,0,:]),dim=-1)

        text = self.tokenizer(caption, padding='max_length', truncation=True,
         ↪  max_length=30,
                              return_tensors="pt").to(image.device)
        text_output = self.text_encoder(text.input_ids, attention_mask =
         ↪  text.attention_mask,
                                        return_dict = True, mode = 'text')
        text_feat =
         ↪  F.normalize(self.text_proj(text_output.last_hidden_state[:,0,:]),dim=-1)

        # get momentum features
        with torch.no_grad():
            self._momentum_update()
            image_embeds_m = self.visual_encoder_m(image)
            image_feat_m = F.normalize(self.vision_proj_m(image_embeds_m[:,0,:]),dim=-1)
            image_feat_all =
            ↪  torch.cat([image_feat_m.t(),self.image_queue.clone().detach()],dim=1)

            text_output_m = self.text_encoder_m(text.input_ids, attention_mask =
            ↪  text.attention_mask,
                                                return_dict = True, mode = 'text')
            text_feat_m =
            ↪  F.normalize(self.text_proj_m(text_output_m.last_hidden_state[:,0,:]),dim=-1)
            text_feat_all =
            ↪  torch.cat([text_feat_m.t(),self.text_queue.clone().detach()],dim=1)

            sim_i2t_m = image_feat_m @ text_feat_all / self.temp
            sim_t2i_m = text_feat_m @ image_feat_all / self.temp

            sim_targets = torch.zeros(sim_i2t_m.size()).to(image.device)
            sim_targets.fill_diagonal_(1)

            sim_i2t_targets = alpha * F.softmax(sim_i2t_m, dim=1) + (1 - alpha) *
            ↪  sim_targets
            sim_t2i_targets = alpha * F.softmax(sim_t2i_m, dim=1) + (1 - alpha) *
            ↪  sim_targets

        sim_i2t = image_feat @ text_feat_all / self.temp
        sim_t2i = text_feat @ image_feat_all / self.temp

        loss_i2t = -torch.sum(F.log_softmax(sim_i2t,
        ↪  dim=1)*sim_i2t_targets,dim=1).mean()
        loss_t2i = -torch.sum(F.log_softmax(sim_t2i,
        ↪  dim=1)*sim_t2i_targets,dim=1).mean()

        loss_ita = (loss_i2t+loss_t2i)/2

        self._dequeue_and_enqueue(image_feat_m, text_feat_m)

        ###=============== Image-text Matching ===================###
        encoder_input_ids = text.input_ids.clone()
        encoder_input_ids[:,0] = self.tokenizer.enc_token_id

        # forward the positve image-text pair
        bs = image.size(0)
        output_pos = self.text_encoder(encoder_input_ids,
                                       attention_mask = text.attention_mask,
                                       encoder_hidden_states = image_embeds,
                                       encoder_attention_mask = image_atts,
                                       return_dict = True,
                                      )
        with torch.no_grad():
            weights_t2i = F.softmax(sim_t2i[:,:bs],dim=1)+1e-4
            weights_t2i.fill_diagonal_(0)
            weights_i2t = F.softmax(sim_i2t[:,:bs],dim=1)+1e-4
            weights_i2t.fill_diagonal_(0)

        # select a negative image for each text
        image_embeds_neg = []
        for b in range(bs):
            neg_idx = torch.multinomial(weights_t2i[b], 1).item()
            image_embeds_neg.append(image_embeds[neg_idx])
        image_embeds_neg = torch.stack(image_embeds_neg,dim=0)

        # select a negative text for each image
        text_ids_neg = []
        text_atts_neg = []
        for b in range(bs):
            neg_idx = torch.multinomial(weights_i2t[b], 1).item()
            text_ids_neg.append(encoder_input_ids[neg_idx])
            text_atts_neg.append(text.attention_mask[neg_idx])

        text_ids_neg = torch.stack(text_ids_neg,dim=0)
        text_atts_neg = torch.stack(text_atts_neg,dim=0)

        text_ids_all = torch.cat([encoder_input_ids, text_ids_neg],dim=0)
        text_atts_all = torch.cat([text.attention_mask, text_atts_neg],dim=0)

        image_embeds_all = torch.cat([image_embeds_neg,image_embeds],dim=0)
        image_atts_all = torch.cat([image_atts,image_atts],dim=0)

        output_neg = self.text_encoder(text_ids_all,
                                       attention_mask = text_atts_all,
                                       encoder_hidden_states = image_embeds_all,
                                       encoder_attention_mask = image_atts_all,
                                       return_dict = True,
                                      )

        vl_embeddings = torch.cat([output_pos.last_hidden_state[:,0,:],
        ↪   output_neg.last_hidden_state[:,0,:]],dim=0)
        vl_output = self.itm_head(vl_embeddings)

        itm_labels =
        ↪   torch.cat([torch.ones(bs,dtype=torch.long),torch.zeros(2*bs,dtype=torch.long)],
                               dim=0).to(image.device)
        loss_itm = F.cross_entropy(vl_output, itm_labels)

        ##================= LM ========================##
        decoder_input_ids = text.input_ids.clone()
        decoder_input_ids[:,0] = self.tokenizer.bos_token_id
        decoder_targets = decoder_input_ids.masked_fill(decoder_input_ids ==
        ↪   self.tokenizer.pad_token_id, -100)

        decoder_output = self.text_decoder(decoder_input_ids,
                                           attention_mask = text.attention_mask,
                                           encoder_hidden_states = image_embeds,
                                           encoder_attention_mask = image_atts,
                                           labels = decoder_targets,
                                           return_dict = True,
                                          )

        loss_lm = decoder_output.loss
        return loss_ita, loss_itm, loss_lm



    @torch.no_grad()
    def copy_params(self):
        for model_pair in self.model_pairs:
            for param, param_m in zip(model_pair[0].parameters(),
            ↪ model_pair[1].parameters()):
                param_m.data.copy_(param.data)  # initialize
                param_m.requires_grad = False  # not update by gradient


    @torch.no_grad()
    def _momentum_update(self):
        for model_pair in self.model_pairs:
            for param, param_m in zip(model_pair[0].parameters(),
            ↪ model_pair[1].parameters()):
                param_m.data = param_m.data * self.momentum + param.data * (1. -
                ↪ self.momentum)


    @torch.no_grad()
    def _dequeue_and_enqueue(self, image_feat, text_feat):
        # gather keys before updating queue
        image_feats = concat_all_gather(image_feat)
        text_feats = concat_all_gather(text_feat)

        batch_size = image_feats.shape[0]

        ptr = int(self.queue_ptr)
        assert self.queue_size % batch_size == 0  # for simplicity

        # replace the keys at ptr (dequeue and enqueue)
        self.image_queue[:, ptr:ptr + batch_size] = image_feats.T
        self.text_queue[:, ptr:ptr + batch_size] = text_feats.T
        ptr = (ptr + batch_size) % self.queue_size  # move pointer

        self.queue_ptr[0] = ptr


def blip_pretrain(**kwargs):
    model = BLIP_Pretrain(**kwargs)
    return model


@torch.no_grad()
```

```python
def concat_all_gather(tensor):
    """
    Performs all_gather operation on the provided tensors.
    *** Warning ***: torch.distributed.all_gather has no gradient.
    """
    tensors_gather = [torch.ones_like(tensor)
        for _ in range(torch.distributed.get_world_size())]
    torch.distributed.all_gather(tensors_gather, tensor, async_op=False)

    output = torch.cat(tensors_gather, dim=0)
    return output


from typing import List
def tie_encoder_decoder_weights(encoder: nn.Module, decoder: nn.Module,
↪   base_model_prefix: str, skip_key:str):
    uninitialized_encoder_weights: List[str] = []
    if decoder.__class__ != encoder.__class__:
        logger.info(
            f"{decoder.__class__} and {encoder.__class__} are not equal. In this case
            ↪  make sure that all encoder weights are correctly initialized."
        )

    def tie_encoder_to_decoder_recursively(
        decoder_pointer: nn.Module,
        encoder_pointer: nn.Module,
        module_name: str,
        uninitialized_encoder_weights: List[str],
        skip_key: str,
        depth=0,
    ):
        assert isinstance(decoder_pointer, nn.Module) and isinstance(
            encoder_pointer, nn.Module
        ), f"{decoder_pointer} and {encoder_pointer} have to be of type torch.nn.Module"
        if hasattr(decoder_pointer, "weight") and skip_key not in module_name:
            assert hasattr(encoder_pointer, "weight")
            encoder_pointer.weight = decoder_pointer.weight
            if hasattr(decoder_pointer, "bias"):
                assert hasattr(encoder_pointer, "bias")
                encoder_pointer.bias = decoder_pointer.bias
                print(module_name+' is tied')
                return

        encoder_modules = encoder_pointer._modules
        decoder_modules = decoder_pointer._modules
        if len(decoder_modules) > 0:
            assert (
                len(encoder_modules) > 0
            ), f"Encoder module {encoder_pointer} does not match decoder module
            ↪  {decoder_pointer}"

            all_encoder_weights = set([module_name + "/" + sub_name for sub_name in
            ↪  encoder_modules.keys()])
            encoder_layer_pos = 0
            for name, module in decoder_modules.items():
                if name.isdigit():
                    encoder_name = str(int(name) + encoder_layer_pos)
                    decoder_name = name
                    if not isinstance(decoder_modules[decoder_name],
                    ↪  type(encoder_modules[encoder_name])) and len(
                        encoder_modules
                    ) != len(decoder_modules):
                        # this can happen if the name corresponds to the position in a
                        ↪  list module list of layers
                        # in this case the decoder has added a cross-attention that the
                        ↪  encoder does not have
                        # thus skip this step and subtract one layer pos from encoder
                        encoder_layer_pos -= 1
                        continue
                elif name not in encoder_modules:
                    continue
                elif depth > 500:
                    raise ValueError(
                        "Max depth of recursive function `tie_encoder_to_decoder`
                        ↪  reached. It seems that there is a circular dependency
                        ↪  between two or more `nn.Modules` of your model."
                    )
                else:
                    decoder_name = encoder_name = name
                tie_encoder_to_decoder_recursively(
                    decoder_modules[decoder_name],
                    encoder_modules[encoder_name],
                    module_name + "/" + name,
                    uninitialized_encoder_weights,
                    skip_key,
                    depth=depth + 1,
                )
                all_encoder_weights.remove(module_name + "/" + encoder_name)

            uninitialized_encoder_weights += list(all_encoder_weights)

    # tie weights recursively
    tie_encoder_to_decoder_recursively(decoder, encoder, base_model_prefix,
    ↪  uninitialized_encoder_weights, skip_key)

```

A.4 Code for Scenario 04: Training The Model

```python

import os
from tqdm import tqdm
import matplotlib.pyplot as plt
from transformers import BlipForConditionalGeneration, Trainer, TrainingArguments
import torch

# Assume custom_dataset is already defined

# Check if CUDA (GPU) is available
if torch.cuda.is_available():
    print("CUDA is available. Using GPU.")
else:
    print("CUDA is not available. Using CPU.")

# Load pre-trained model
model =
↪   BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-large")

# Define training arguments
training_args = TrainingArguments(
    output_dir="C:/Users/DELL/Desktop/check",
    num_train_epochs=10,
    per_device_train_batch_size=2,
    save_steps=1000,
    save_total_limit=3,
)

# Initialize the Trainer with CUDA for GPU acceleration
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=custom_dataset,
    callbacks=[],
)

# Lists to store training statistics
```

```
train_loss_list = []
train_accuracy_list = []

# Fine-tune the model with tqdm for progress bars and Matplotlib for loss and accuracy
↪  plots
with tqdm(total=trainer.args.num_train_epochs, desc="Training", dynamic_ncols=True) as
↪  pbar:
    for epoch in range(trainer.args.num_train_epochs):


        # Training step
        train_loss = trainer.train().metrics["train_loss"]
        train_accuracy = trainer.train().metrics["train_runtime"]

        train_loss_list.append(train_loss)
        train_accuracy_list.append(train_accuracy)

        # Plot loss and accuracy curves
        plt.figure(figsize=(12, 6))

        plt.subplot(1, 2, 1)
        plt.plot(range(1, epoch + 2), train_loss_list, label='Training Loss')
        plt.title('Training Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.plot(range(1, epoch + 2), train_accuracy_list, label='Training Accuracy')
        plt.title('Training Accuracy')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()

        plt.tight_layout()
        plt.show()

        pbar.update(1)
trainer.train()
# Save the fine-tuned model
```

```
output_model_dir = "C:/Users/DELL/Desktop/check"
trainer.save_model(output_model_dir)
#model.save_weights(os.path.join(output_model_dir, "model_weights.pth"))

```

A.5 Code for Scenario 05: Creating a Caption Generation File

```python

from transformers import BlipForConditionalGeneration, BlipProcessor
from PIL import Image
import torch
import matplotlib.pyplot as plt

# Load the fine-tuned model
model = BlipForConditionalGeneration.from_pretrained("/content/fine_tuned_blip_model")

# Create a Blip processor
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-large")

# Load and preprocess a new image
image_path = "/content/download.jpg"
image = Image.open(image_path).convert("RGB")

# Specify the maximum caption length
max_length = 32

# Generate a caption for the new image
inputs = processor(image, return_tensors="pt", padding="max_length",
↪   max_length=max_length, truncation=True)
outputs = model.generate(**inputs)

# Decode the generated caption
generated_caption = processor.decode(outputs[0], skip_special_tokens=True)

# Print the generated caption
print("Generated Caption:", generated_caption)

plt.show()

```

A.6 Code for Scenario 06:Model Evaluation

```

from transformers import BlipForConditionalGeneration, BlipProcessor
from PIL import Image
from nltk.translate.bleu_score import corpus_bleu, SmoothingFunction
pip install rouge
from rouge import Rouge
import matplotlib.pyplot as plt

# Load the fine-tuned model
model = BlipForConditionalGeneration.from_pretrained("/content/drive/MyDrive/Final
↪ Model")

# Create a Blip processor
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-large")

# Load and preprocess a new image
image_path = "/content/drive/MyDrive/Ammad Ka Dataset/1st person converted/man and
↪ womanjpg500/man_woman_019.jpg"
image = Image.open(image_path).convert("RGB")

# Specify the maximum caption length
max_length = 32

# Generate a caption for the new image
inputs = processor(image, return_tensors="pt", padding="max_length",
↪ max_length=max_length, truncation=True)
outputs = model.generate(**inputs)

# Decode the generated caption
generated_caption = processor.decode(outputs[0], skip_special_tokens=True)

# Print the generated caption
print("Generated Caption:", generated_caption)

# Reference captions (replace with your actual reference captions)
reference_captions = ["I am watching a man holding a child in his arms", "I am looking
↪ at a man holding a baby while looking at the phone", 'There is a man holding a
↪ baby']
```

```python

# BLEU evaluation
def evaluate_bleu(reference, candidate):
    smoothing = SmoothingFunction().method1
    return corpus_bleu([reference], [candidate], smoothing_function=smoothing)

# Rouge evaluation
def evaluate_rouge(reference, candidate):
    rouge = Rouge()
    scores = rouge.get_scores(candidate, reference)
    return scores[0]['rouge-1']['f'], scores[0]['rouge-2']['f'],
    ↪  scores[0]['rouge-l']['f']

# BLEU and Rouge evaluation
for reference_caption in reference_captions:
    # BLEU score
    bleu_score = evaluate_bleu(reference_caption, generated_caption)
    print(f"BLEU Score: {bleu_score}")

    # Rouge scores
    rouge_1, rouge_2, rouge_l = evaluate_rouge(reference_caption, generated_caption)
    print(f"Rouge-1 Score: {rouge_1}")
    print(f"Rouge-2 Score: {rouge_2}")
    print(f"Rouge-L Score: {rouge_l}")


plt.imshow(image)
plt.title("Generated Caption: " + generated_caption)
plt.axis("off")
plt.show()
```

A.7 Code for Scenario 07: Graphical User Interface with Sound Implementation

```python

from tkinter import *
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
from transformers import BlipForConditionalGeneration, BlipProcessor
import pyttsx3
from ttkthemes import ThemedStyle
import cv2
import numpy as np
import sounddevice as sd
import soundfile as sf

# Load the fine-tuned model
model_path ="C:/Users/DELL/Desktop/finetuned 8"
model = BlipForConditionalGeneration.from_pretrained(model_path)

# Create a Blip processor
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-large")

def generate_caption(image_path):
    # Load and preprocess the image
    image = Image.open(image_path).convert("RGB")
    inputs = processor(image, return_tensors="pt", padding="max_length", max_length=32,
    ↪   truncation=True)

    # Generate caption
    outputs = model.generate(**inputs)
    generated_caption = processor.decode(outputs[0], skip_special_tokens=True)

    return generated_caption

def generate_caption_from_pil(pil_image):
    # Preprocess the PIL image
    inputs = processor(pil_image, return_tensors="pt", padding="max_length",
    ↪   max_length=32, truncation=True)

    # Generate caption
    outputs = model.generate(**inputs)
    generated_caption = processor.decode(outputs[0], skip_special_tokens=True)

    return generated_caption

def generate_audio_and_play(caption):
    # Play the audio directly using pyttsx3
    engine = pyttsx3.init()
    engine.say(caption)
    engine.runAndWait()

def choose_file():
    filename = filedialog.askopenfilename(initialdir=".", title="Select image file",
                                          filetypes=(("JPG File", ".jpg"), ("PNG file",
                                          ↪ ".png"), ("All files", "*.*")))
    entry.delete(0, 'end')
    entry.insert(0, filename)

    # Display the selected image
    display_image(filename)

def generate_caption_and_display():
    file_name = entry.get()
    if not file_name:
        messagebox.showerror("Error", "No file selected")
        return

    # Generate caption and display it
    caption = generate_caption(file_name)
    caption_label.config(text="Generated Caption: " + caption)

def display_image(image_path):
    img = Image.open(image_path)
    img = img.resize((500, 500), Image.LANCZOS)
    img = ImageTk.PhotoImage(img)
    image_label.configure(image=img)
    image_label.image = img

def capture_live_image():
    cap = cv2.VideoCapture(0)  # Open the default camera (change to 1 if you have an
    ↪  external camera)

    # Set up audio capture
    audio_fs = 44100  # Audio sampling rate
    audio_channels = 2  # Stereo
    audio_duration = 2  # Duration of audio to capture (in seconds)

    while True:
        ret, frame = cap.read()

        if not ret:
            messagebox.showerror("Error", "Unable to capture live image")
            break

        # Convert OpenCV image (BGR) to PIL image (RGB)
        pil_image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))

        # Display the live image in the GUI
        display_image_pil(pil_image)

        # Generate and display the live caption
        live_caption = generate_caption_from_pil(pil_image)
        caption_label.config(text="Live Caption: " + live_caption)

        # Play the audio of the live caption
        generate_audio_and_play(live_caption)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

def display_image_pil(pil_image):
    # Resize the image for display
    pil_image = pil_image.resize((500, 500), Image.LANCZOS)
    img = ImageTk.PhotoImage(pil_image)
    image_label.configure(image=img)
    image_label.image = img
    root.update_idletasks()  # Update the GUI to display the live image

```

```python
# Main Tkinter window
root = Tk()
root.title("Image Captioning Application")
root.geometry("800x600")
root.configure(bg="#000000")  # Set overall background color to black

# Apply a dark theme
style = ThemedStyle(root)
style.set_theme("equilux")

# Add two images to the left and right
left_image = Image.open("C:/Users/DELL/Desktop/caea.png")
left_image = left_image.resize((275, 300))
left_image = ImageTk.PhotoImage(left_image)
left_label = Label(root, image=left_image, bg="#000000")
left_label.pack(side=LEFT, padx=10, pady=10, anchor=N)

right_image = Image.open("C:/Users/DELL/Desktop/nusta.png")
right_image = right_image.resize((300, 300))
right_image = ImageTk.PhotoImage(right_image)
right_label = Label(root, image=right_image, bg="#000000")
right_label.pack(side=RIGHT, padx=10, pady=10, anchor=N)

# Create and place widgets
frame = Frame(root, padx=20, pady=20, bg="#000000")  # Set frame background color to
↪  black
frame.pack(expand=True, fill="both")

# Add a stylish label at the top
title_label = Label(frame, text="Image Captioning App", font=("Helvetica", 24, "bold"),
↪  fg="#FFFFFF", bg="#000000")  # Set text and foreground color
title_label.pack(pady=10)

entry = Entry(frame, width=70, font=("Arial", 16), bg="#111111", fg="#FFFFFF")  # Set
↪  entry background color and text color
entry.pack(pady=10)

browse_button = Button(frame, text="Browse", command=choose_file, bg="#4CAF50",
↪  fg="white", font=("Arial", 16))  # Set button color
browse_button.pack(pady=10)
```

```

generate_button = Button(frame, text="Generate Caption",
↪   command=generate_caption_and_display, bg="#3498DB", fg="white", font=("Arial", 16))
↪   # Set button color
generate_button.pack(pady=10)

generate_audio_button = Button(frame, text="Generate Audio", command=lambda:
↪   generate_audio_and_play(caption_label.cget("text")[18:]), bg="#E74C3C", fg="white",
↪   font=("Arial", 16))  # Set button color
generate_audio_button.pack(pady=10)

live_capture_button = Button(frame, text="Live Capture", command=capture_live_image,
↪   bg="#FFD700", fg="black", font=("Arial", 16))  # Set button color
live_capture_button.pack(pady=10)

image_label = Label(frame, bg="#000000")  # Set image label background color to black
image_label.pack(pady=10)

caption_label = Label(frame, text="Generated Caption: ", font=("Arial", 16),
↪   bg="#000000", fg="#FFFFFF")  # Set caption label background color and text color
caption_label.pack(pady=10)

root.mainloop()

```

Bibliography

[1] Plizzari, C., Goletto, G., Furnari, A., Bansal, S., Ragusa, F., Farinella, G. M., Tommasi, T. (2023). An outlook into the future of egocentric vision. arXiv preprint arXiv:2308.07123.

[2] S. Sathe, S. Shinde, S. Chorge, S. Thakare, and L. Kulkarni, "Overview of image caption generators and its applications," pp. 105–110, 2022.

[3] K. Delloul and S. Larabi, "Egocentric Scene Description for the Blind and Visually Impaired," in Proceedings of the 5th International Symposium on Informatics and its Applications (ISIA), M'sila, Algeria, 2022, pp. 1-6, doi: 10.1109/ISIA55826.2022.9993531.

[4] S. Bai and S. An, "A survey on automatic image caption generation," Neurocomputing, vol. 311, pp. 291304, 10 2018.

[5] M. D. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "A comprehensive survey of deep learning for image captioning," ACM Computing Surveys (CSUR), vol. 51, 2 2019.

[6] S. Liu, L. Bai, Y. Hu, and H. Wang, "Image captioning based on deep neural networks," vol. 232, EDP Sciences, 11 2018.

[7] Z. Ren, X. Wang, N. Zhang, X. Lv, and L.-J. Li, "Deep reinforcement learningbased image captioning with embedding reward," 2017.

[8] J. Chen, H. Guo, K. Yi, B. Li, and M. Elhoseiny, "Visualgpt: Data-efficient adaptation of pretrained language models for image captioning,"

[9] N. Aafaq, N. Akhtar, W. Liu, and A. Mian, "Empirical autopsy of deep video captioning encoder-decoder architecture," Array, vol. 9, p. 100052, 3 2021.

[10] M. A. Al-Malla, A. Jafar, and N. Ghneim, "Image captioning model using attention and object features to mimic human image understanding," Journal of Big Data, vol. 9, 12 2022.

[11] N. BM, “What is an encoder decoder model? — by nechu bm — towards data science.

[12] A. Nursikuwagus, R. Munir, and M. L. Khodra, “Hybrid of deep learning and word embedding in generating captions: Image-captioning solution for geological rock images,” Journal of Imaging, vol. 8, 11 2022.

[13] V. Atliha and D. Seˇ sok, “Pretrained word embeddings for image captioning,” in ˇ 2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), pp. 1–4, 2021.

[14] A. Gupta, “A comprehensive guide on deep learning optimizers,” 10 2021.

[15] Musstafa, “Optimizers in deep learning. what is an optimizer? — by musstafa — mlearning.ai — medium,” 3 2021.

[16] D. Gupta, “Activation functions — fundamentals of deep learning,” 12 2022.

[17] P. Marimuthu, “How activation functions work in deep learning - kdnuggets,” 6 2022.

[18] C. Callison-Burch, M. Osborne, and P. Koehn, ”Re-evaluating the role of BLEU in machine translation research,” in Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics, 2006, pp. 249-256.

[19] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, ”Bleu: a method for automatic evaluation of machine translation,” in Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, 2002, pp. 311-318.

[20] A. Lavie and M. J. Denkowski, ”The METEOR metric for automatic evaluation of machine translation,” Machine Translation, vol. 23, pp. 105-115, 2009.

[21] M. Denkowski and A. Lavie, ”Meteor universal: Language specific translation evaluation for any target language,” in Proceedings of the Ninth Workshop on Statistical Machine Translation, 2014, pp. 376-380.

[22] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, ”Cider: Consensus based image description evaluation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 4566-4575.

[23] P. Anderson, B. Fernando, M. Johnson, and S. Gould, ”Spice: Semantic propositional image caption evaluation,” in Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part V, vol. 14, pp. 382-398, Springer International Publishing.

[24] C. Y. Lin, ”Rouge: A package for automatic evaluation of summaries,” in Text Summarization Branches Out, 2004, pp. 74-81.

[25] M. Kilickaya, A. Erdem, N. Ikizler-Cinbis, and E. Erdem, ”Reevaluating automatic metrics for image captioning,” arXiv preprint arXiv:1612.07600, 2016.

[26] Smith, J., Johnson, A. (2023). Annotation Strategies for Enriching Image Datasets. Journal of Computer Vision.

[27] N. Aafaq, “Deep learning for natural language description of videos,” 2021.

[28] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, ”Natural Language Processing (Almost) from Scratch,” Journal of Machine Learning Research, vol. 12, 2011, pp. 2493-2537.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” arXiv preprint arXiv:1810.04805, 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, ”Efficient Estimation of Word Representations in Vector Space,” arXiv preprint arXiv:1301.3781, 2013.

[31] J. Howard and S. Ruder, ”Universal Language Model Fine-tuning for Text Classification,” *arXiv preprint arXiv:1801.06146*, 2018. [Online]. Available: https://arxiv.org/abs/1801.06146

[32] J. Brownlee, “How to develop a deep learning photo caption generator from scratch - machinelearningmastery.com,” 12 2020